Pandas Visualization

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

%matplotlib notebook
```

In [2]:
```
# see the pre-defined styles provided.
plt.style.available
```

Out[2]:
```
['seaborn-dark',
 'seaborn-colorblind',
 'bmh',
 'dark_background',
 'seaborn-ticks',
 'seaborn-paper',
 'seaborn',
 'seaborn-poster',
 'seaborn-notebook',
 'fivethirtyeight',
 'seaborn-pastel',
 'seaborn-dark-palette',
 'grayscale',
 'seaborn-talk',
 'seaborn-deep',
 'seaborn-bright',
 'ggplot',
 'seaborn-whitegrid',
 'seaborn-white',
 'seaborn-muted',
 'seaborn-darkgrid',
 'classic']
```

In [3]:
```
# use the 'seaborn-colorblind' style
plt.style.use('seaborn-colorblind')
```

DataFrame.plot

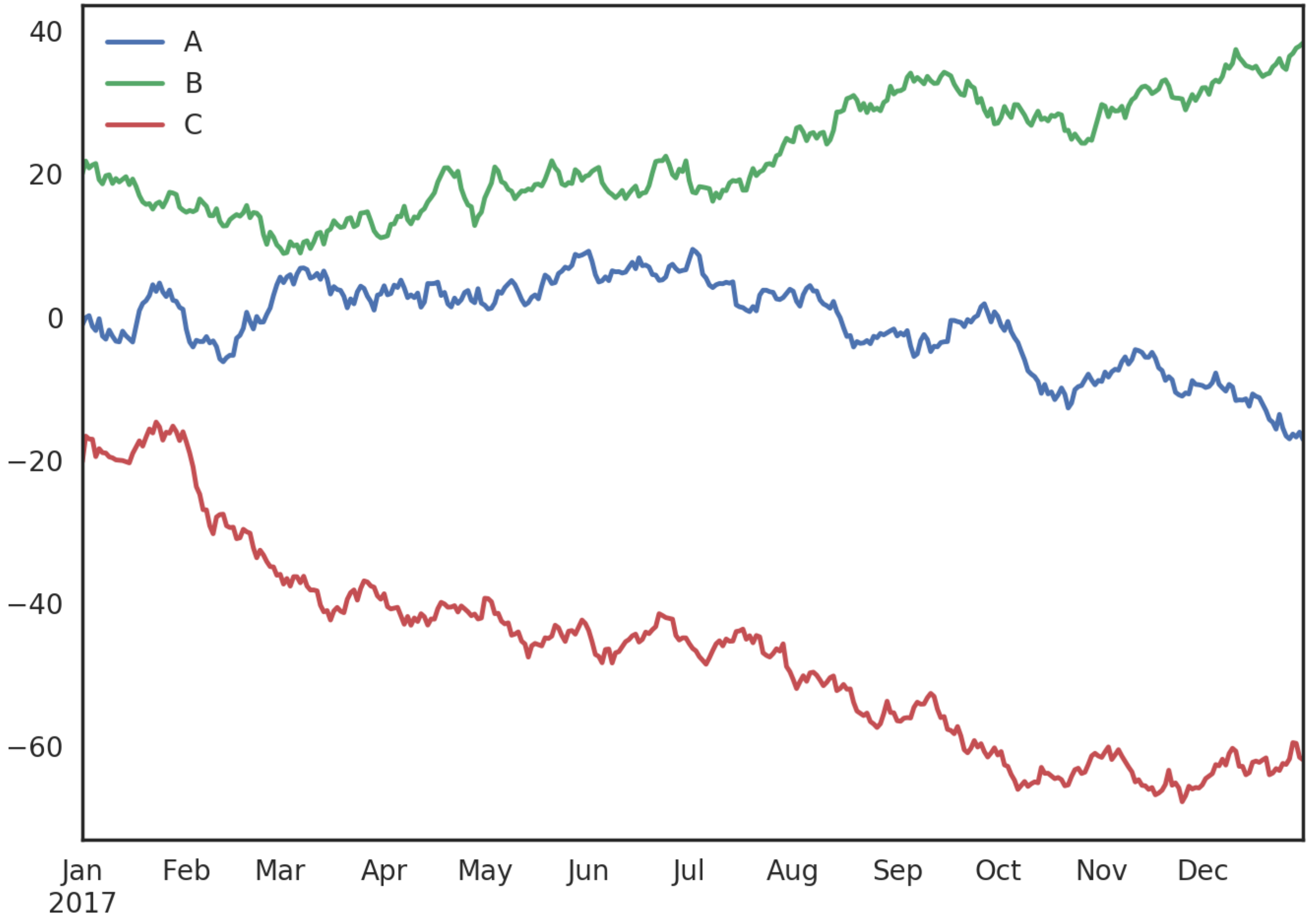
In [4]:
```
np.random.seed(123)

df = pd.DataFrame({'A': np.random.randn(365).cumsum(0),
                   'B': np.random.randn(365).cumsum(0) + 20,
                   'C': np.random.randn(365).cumsum(0) - 20},
                  index=pd.date_range('1/1/2017', periods=365))
df.head()
```

Out[4]:

	A	B	C
2017-01-01	-1.085631	20.059291	-20.230904
2017-01-02	-0.088285	21.803332	-16.659325
2017-01-03	0.194693	20.835588	-17.055481
2017-01-04	-1.311601	21.255156	-17.093802
2017-01-05	-1.890202	21.462083	-19.518638

```
In [27]: df.plot(); # add a semi-colon to the end of the plotting call to suppress unwanted output
```

Figure 10

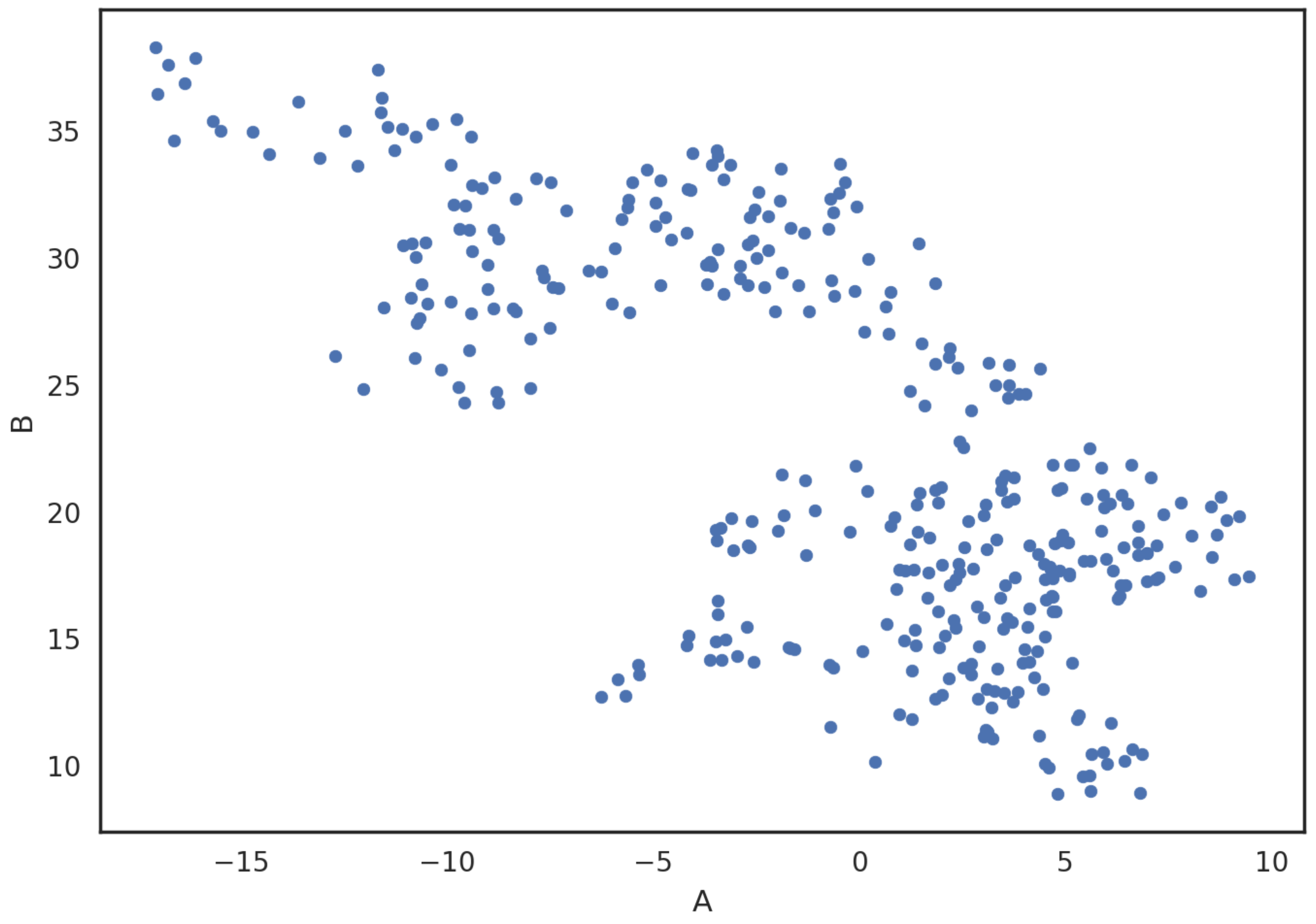


Back to previous view

We can select which plot we want to use by passing it into the 'kind' parameter.

```
In [28]: df.plot('A','B', kind = 'scatter');
```

Figure 11



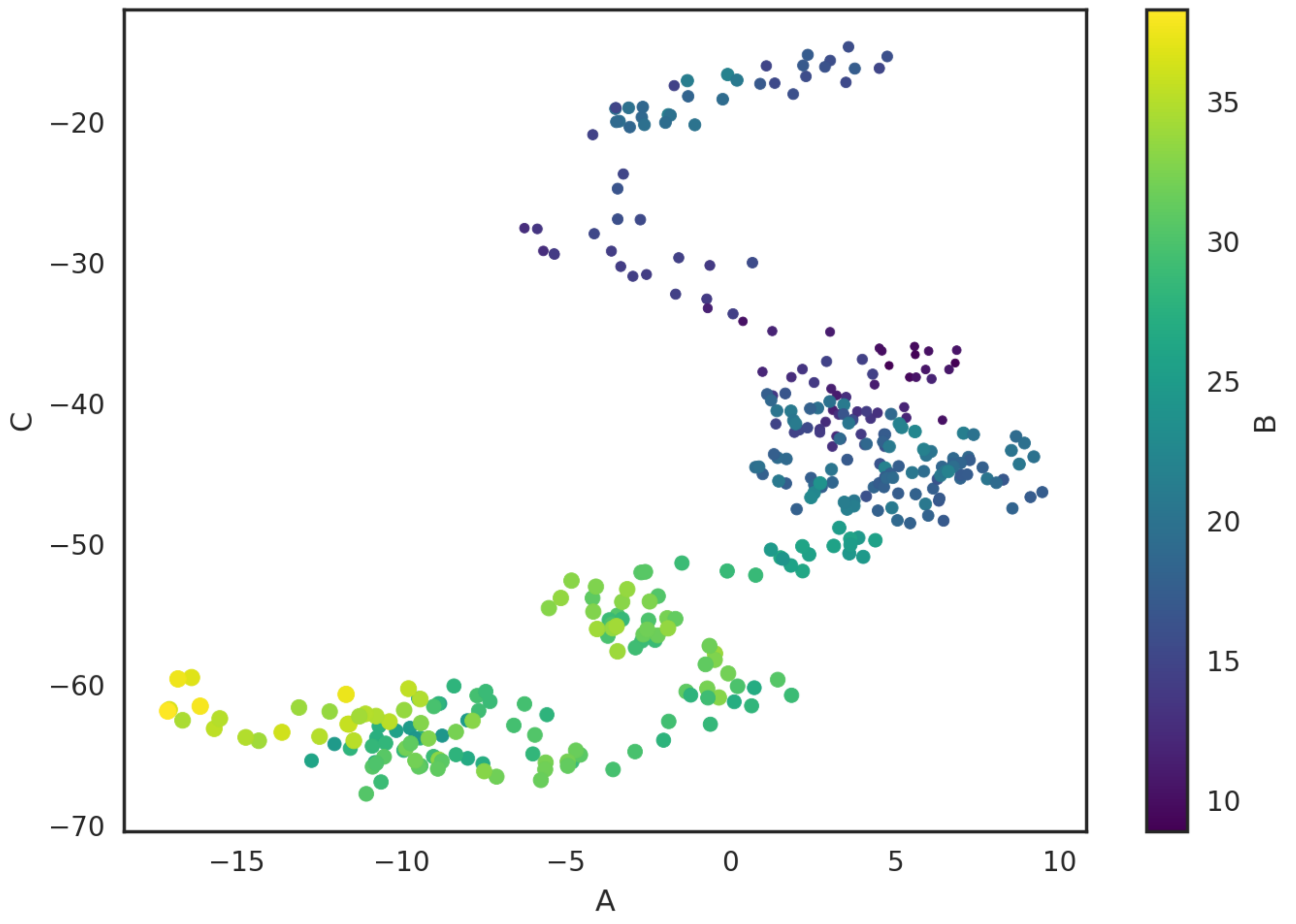
You can also choose the plot kind by using the `DataFrame.plot.kind` methods instead of providing the `kind` keyword argument.

`kind` :

- `'line'` : line plot (default)
- `'bar'` : vertical bar plot
- `'barh'` : horizontal bar plot
- `'hist'` : histogram
- `'box'` : boxplot
- `'kde'` : Kernel Density Estimation plot
- `'density'` : same as 'kde'
- `'area'` : area plot
- `'pie'` : pie plot
- `'scatter'` : scatter plot
- `'hexbin'` : hexbin plot

```
In [29]: # create a scatter plot of columns 'A' and 'C', with changing color (c) and size (s) based on column 'B'
         df.plot.scatter('A', 'C', c='B', s=df['B'], colormap='viridis')
```

Figure 12

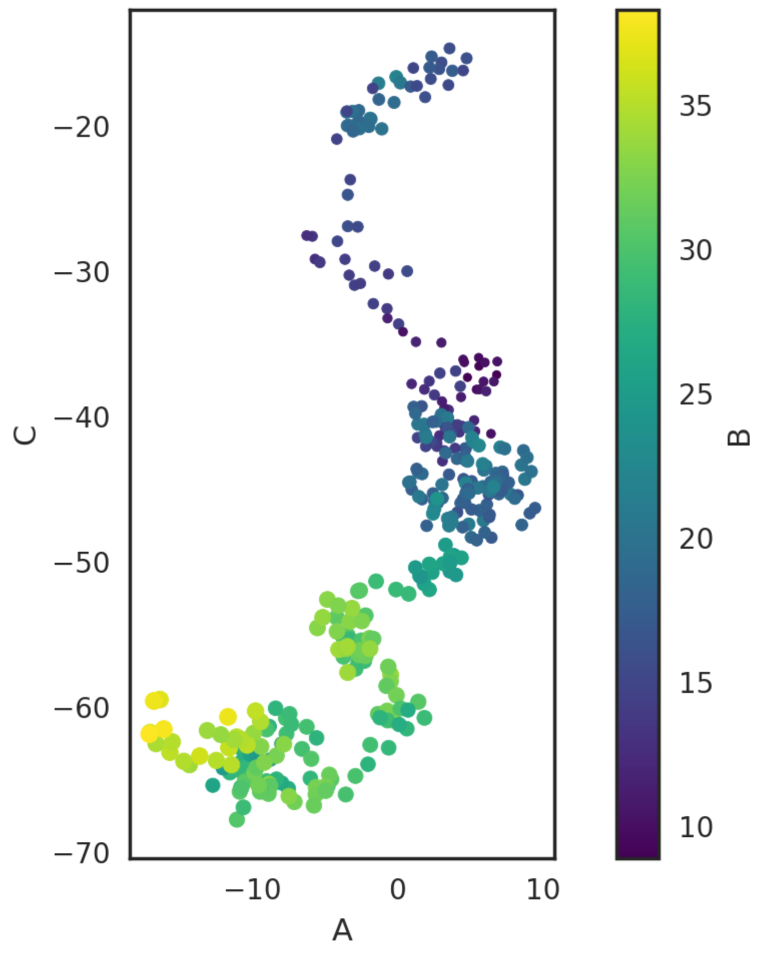


Forward to next view

```
Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcb3f180438>
```

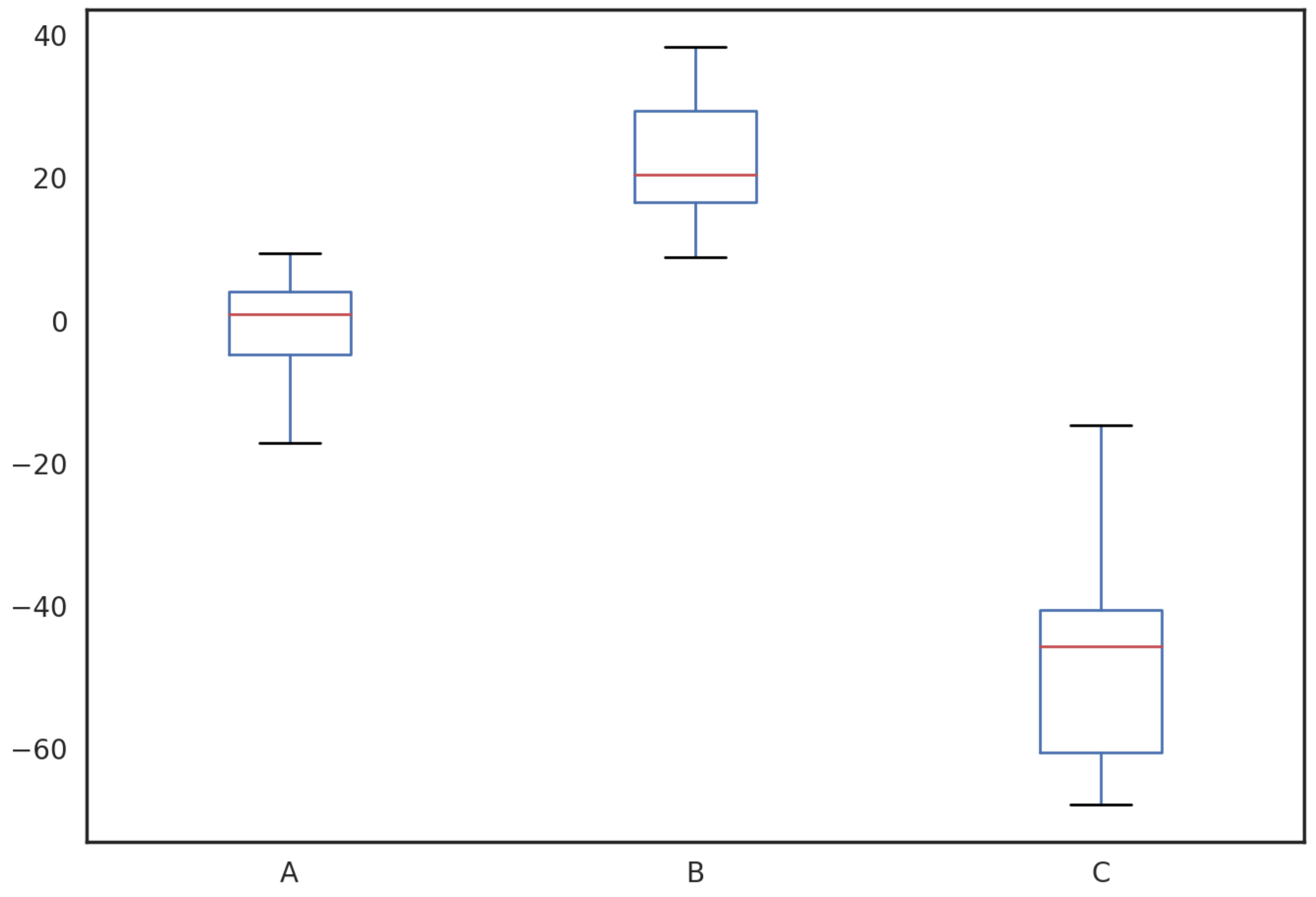
```
In [30]: ax = df.plot.scatter('A', 'C', c='B', s=df['B'], colormap='viridis')
         ax.set_aspect('equal')
```

Figure 13



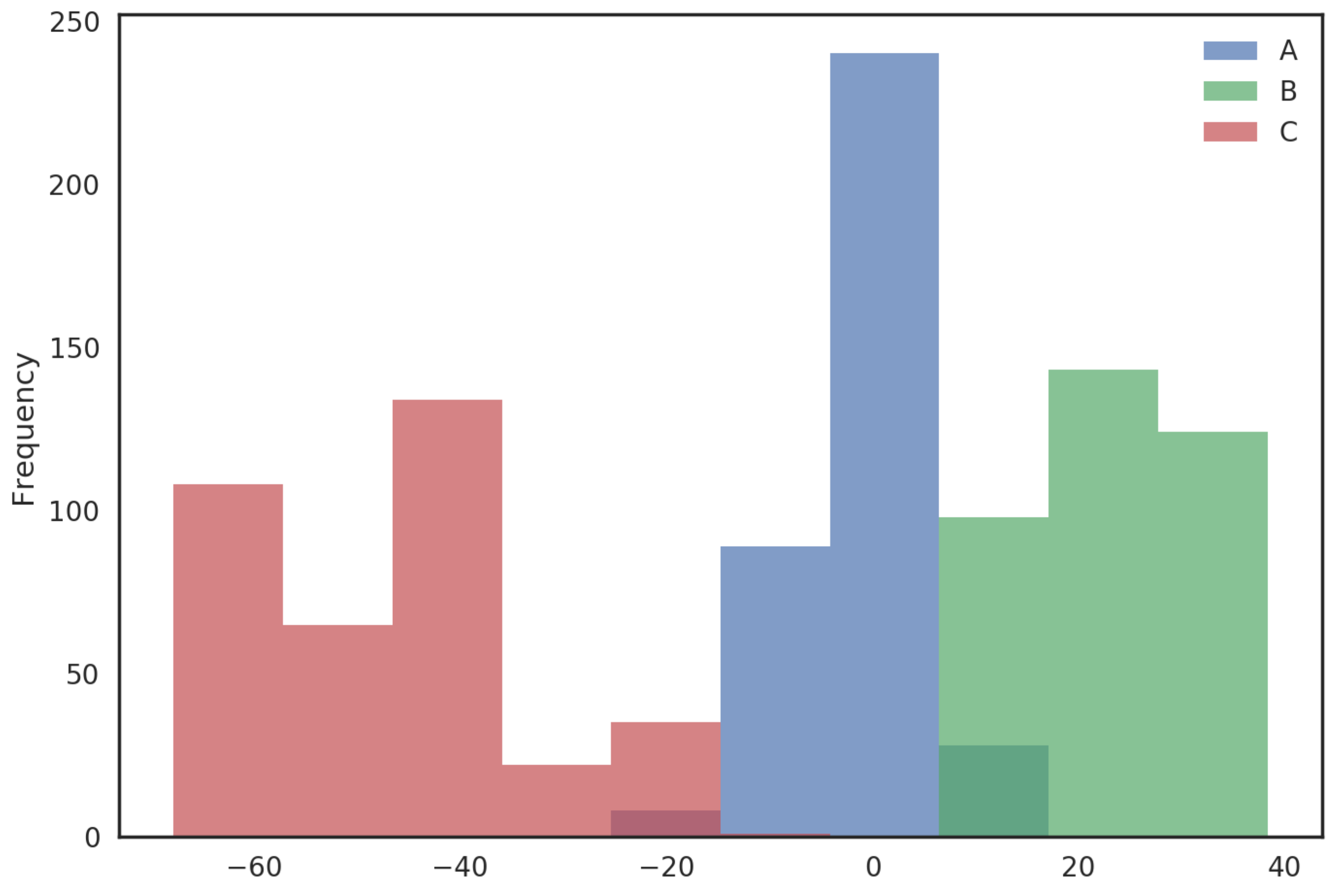
In [31]:
```
df.plot.box();
```

Figure 14



In [32]:
```
df.plot.hist(alpha=0.7);
```

Figure 15

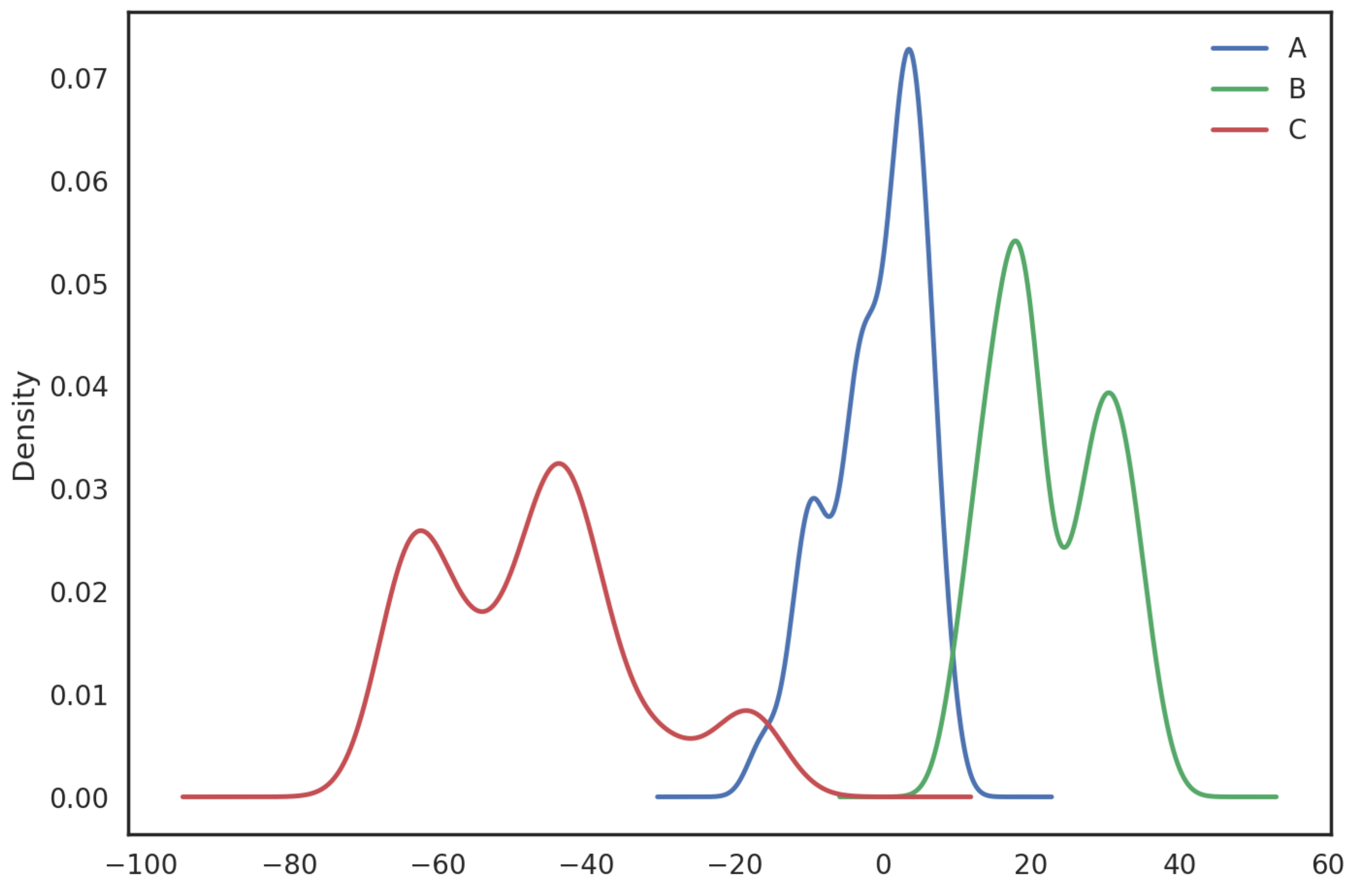


Back to previous view

Kernel density estimation plots (https://en.wikipedia.org/wiki/Kernel_density_estimation) are useful for deriving a smooth continuous function from a given sample.

In [33]:
```
df.plot.kde();
```

Figure 16



pandas.tools.plotting

Iris flower data set (https://en.wikipedia.org/wiki/Iris_flower_data_set)

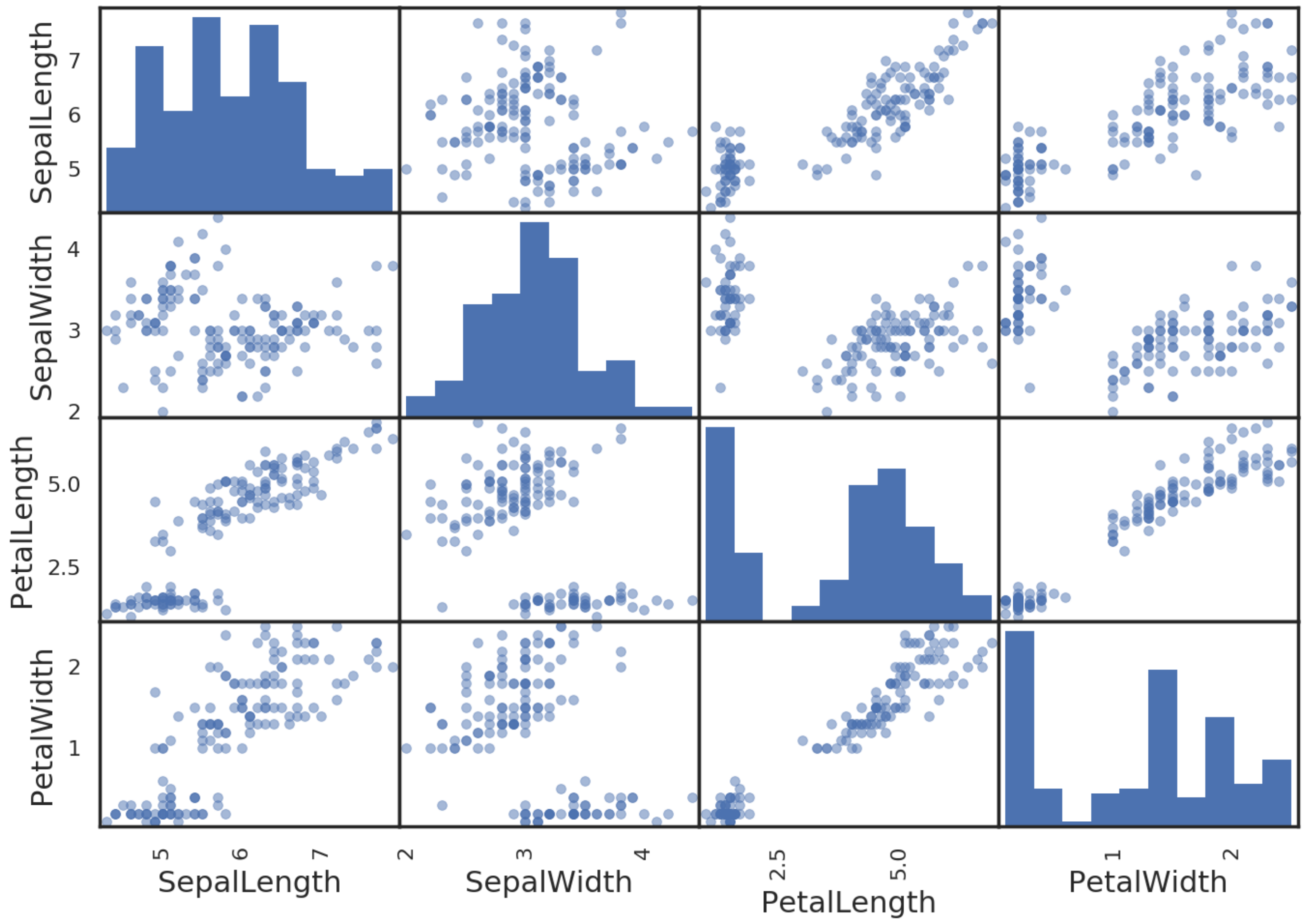
In [12]:
```
iris = pd.read_csv('iris.csv')
iris.head()
```

Out[12]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Name
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
In [34]: pd.tools.plotting.scatter_matrix(iris);
```

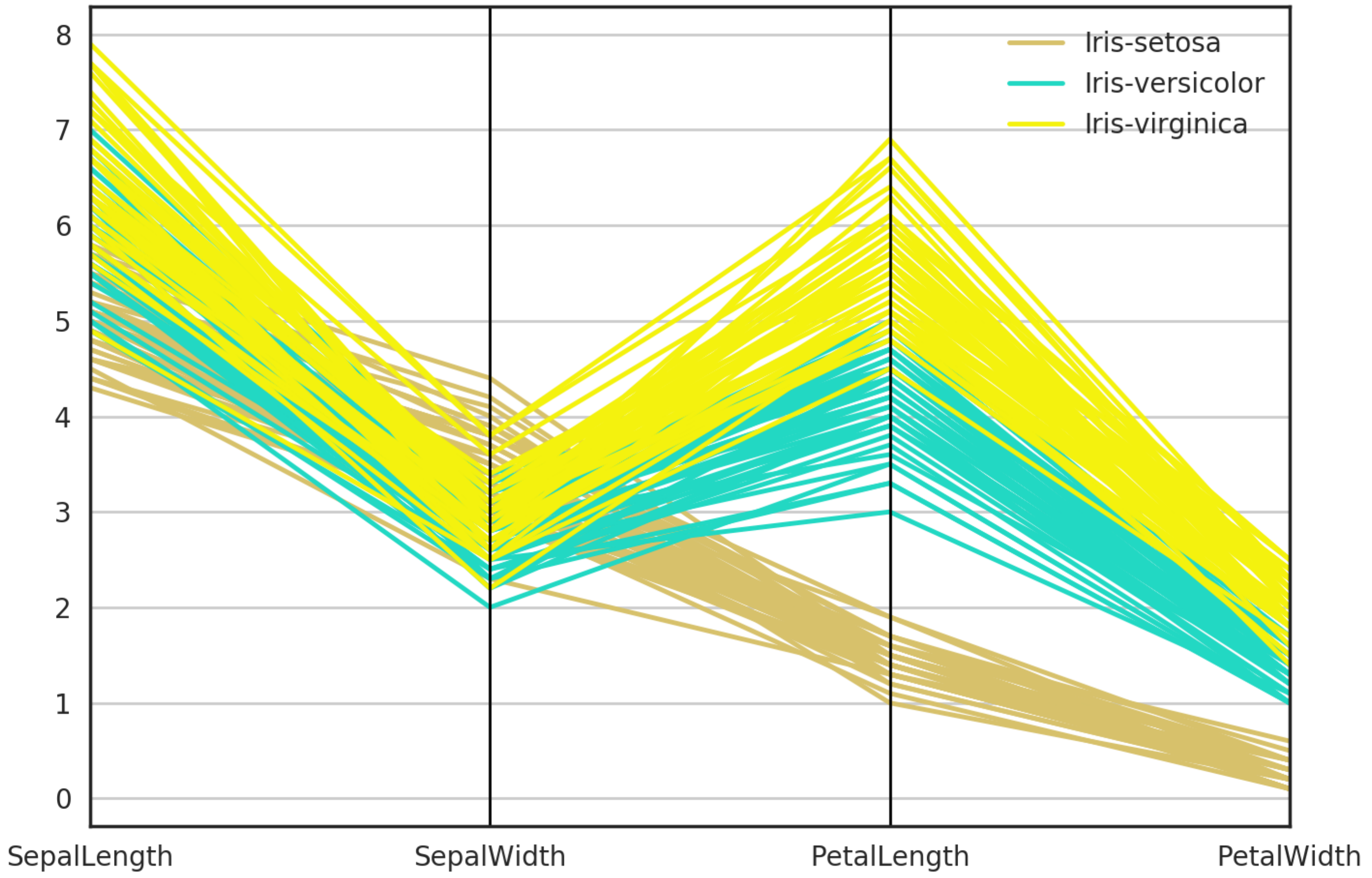
Figure 17



Reset original view

```
In [35]: plt.figure()
         pd.tools.plotting.parallel_coordinates(iris, 'Name');
```

Figure 18



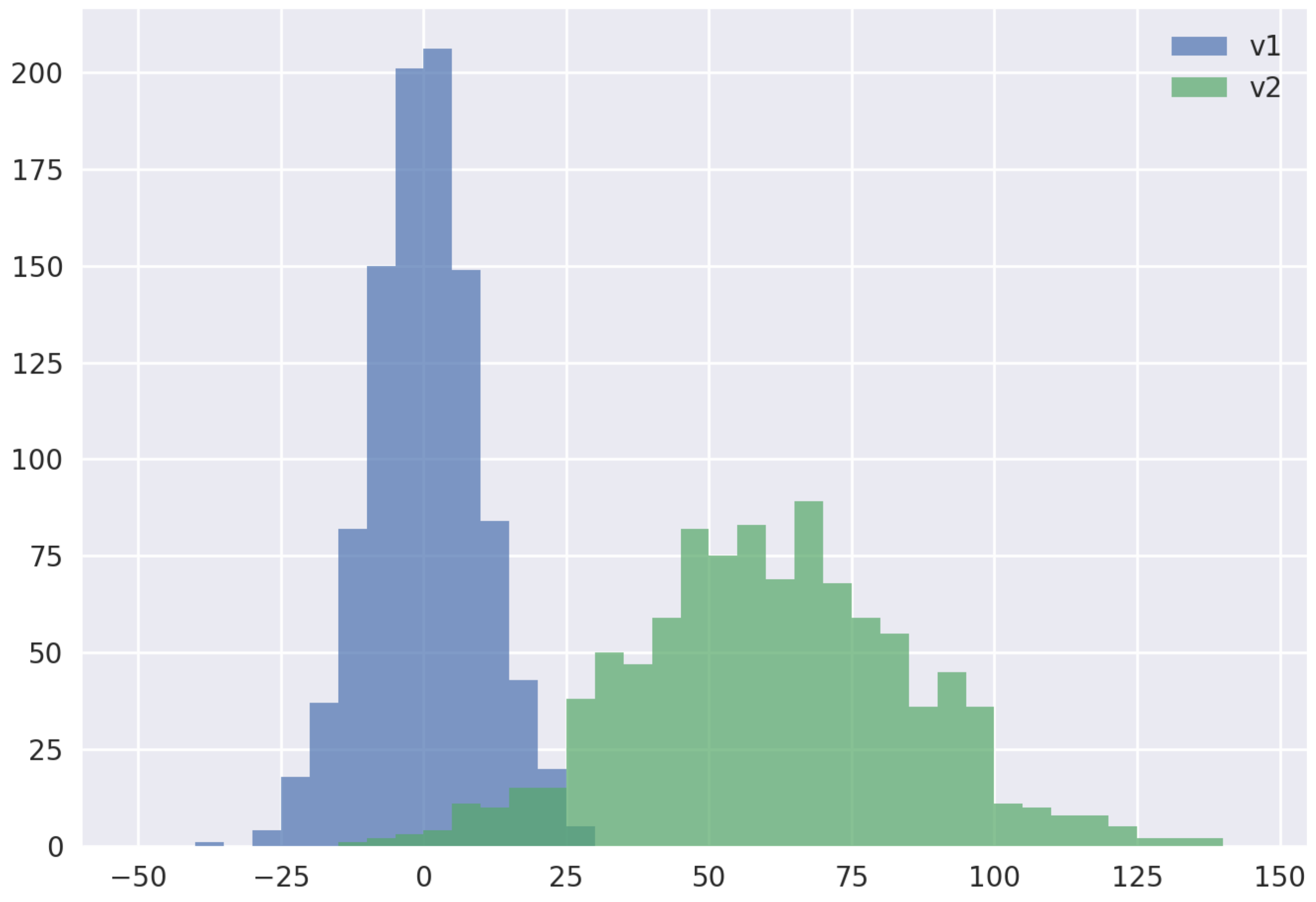
Seaborn

In [15]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib notebook
```

In [16]:
```
np.random.seed(1234)

v1 = pd.Series(np.random.normal(0,10,1000), name='v1')
v2 = pd.Series(2*v1 + np.random.normal(60,15,1000), name='v2')
```

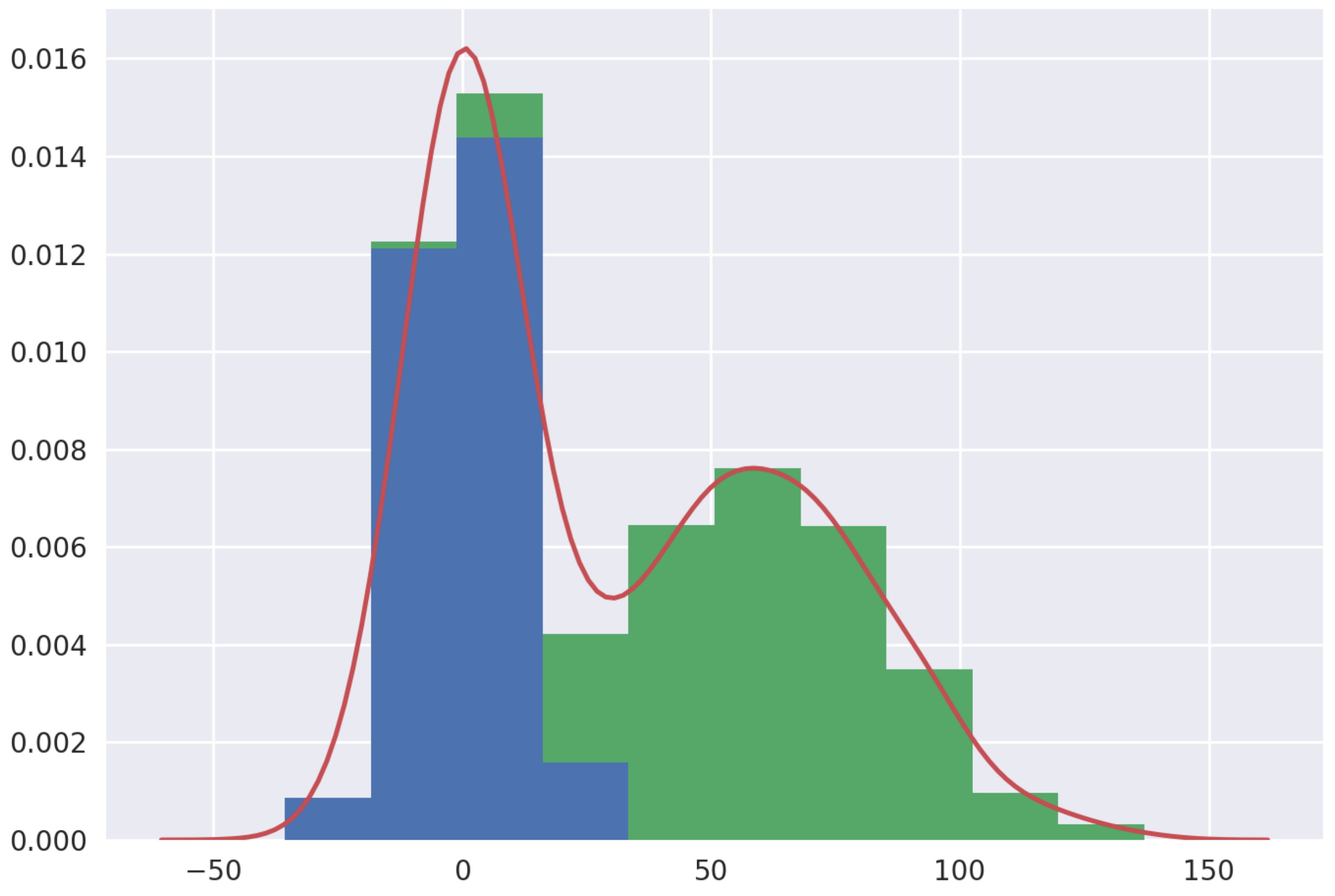
```
In [17]: plt.figure()
         plt.hist(v1, alpha=0.7, bins=np.arange(-50,150,5), label='v1');
         plt.hist(v2, alpha=0.7, bins=np.arange(-50,150,5), label='v2');
         plt.legend();
```

Figure 1



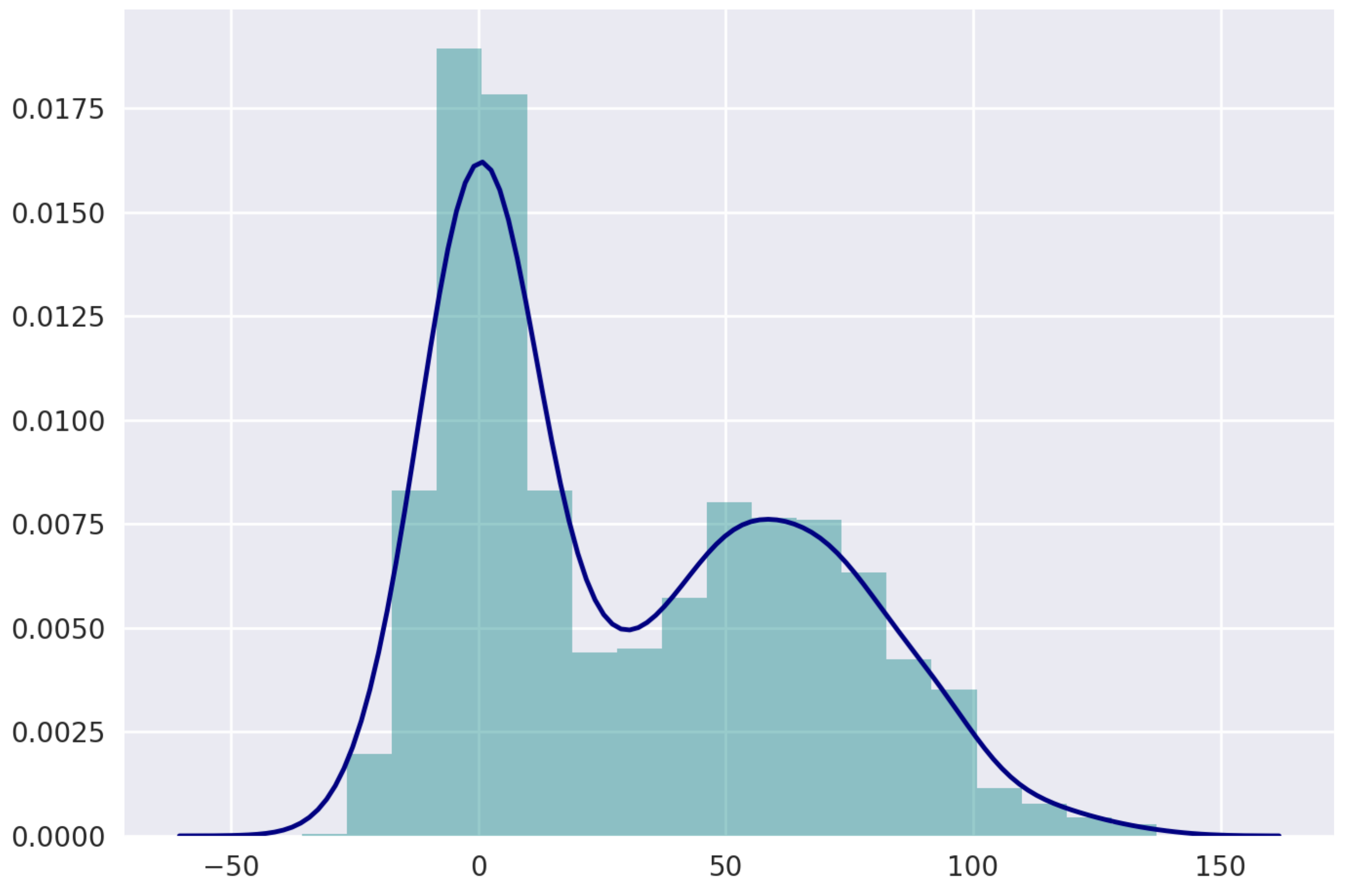
In [18]:
```
# plot a kernel density estimation over a stacked barchart
plt.figure()
plt.hist([v1, v2], histtype='barstacked', normed=True);
v3 = np.concatenate((v1,v2))
sns.kdeplot(v3);
```

Figure 2



```
In [19]: plt.figure()
         # we can pass keyword arguments for each individual component of the plot
         sns.distplot(v3, hist_kws={'color': 'Teal'}, kde_kws={'color': 'Navy'});
```

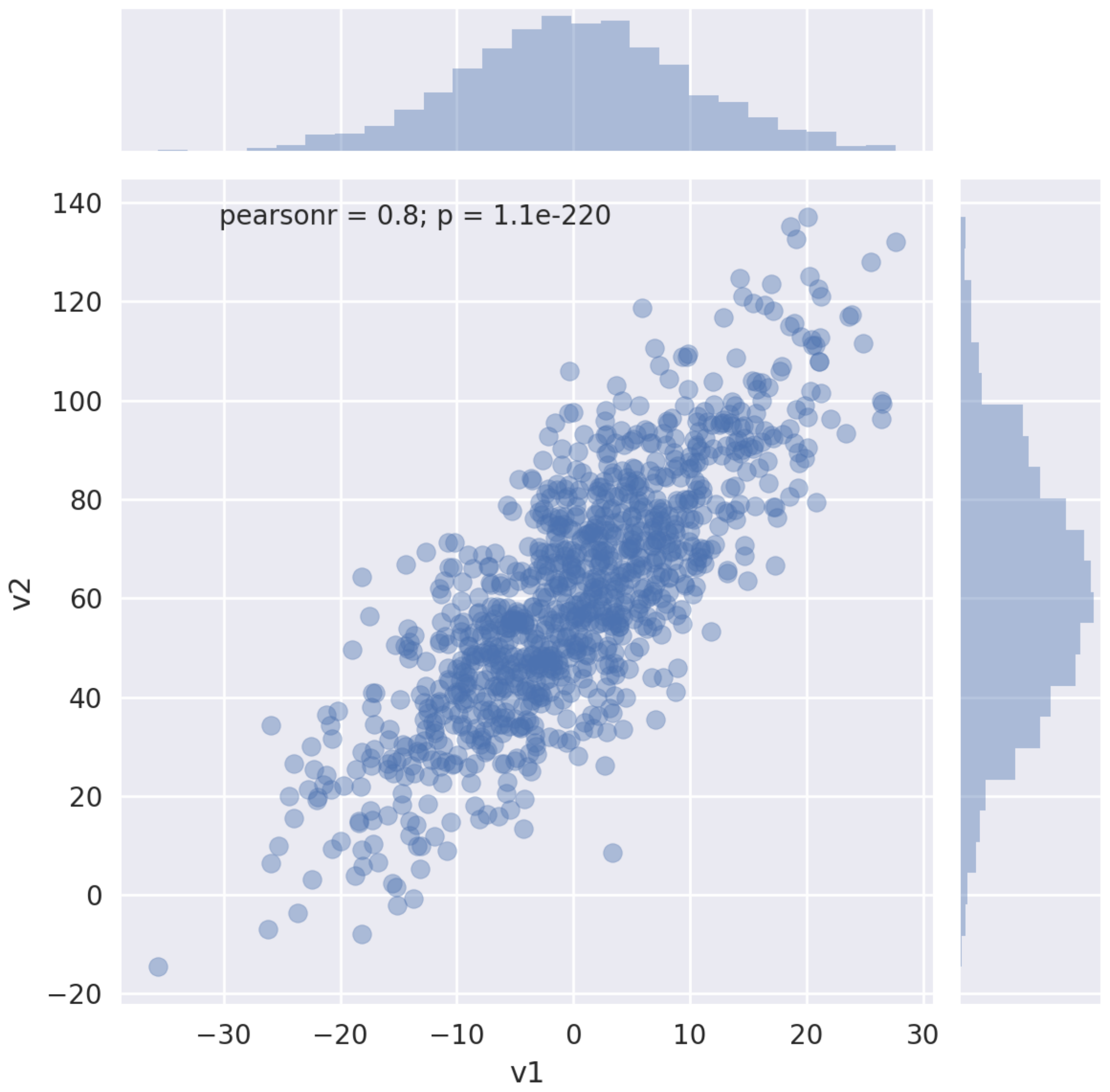
Figure 3



x=92.3085 y=0.00658793

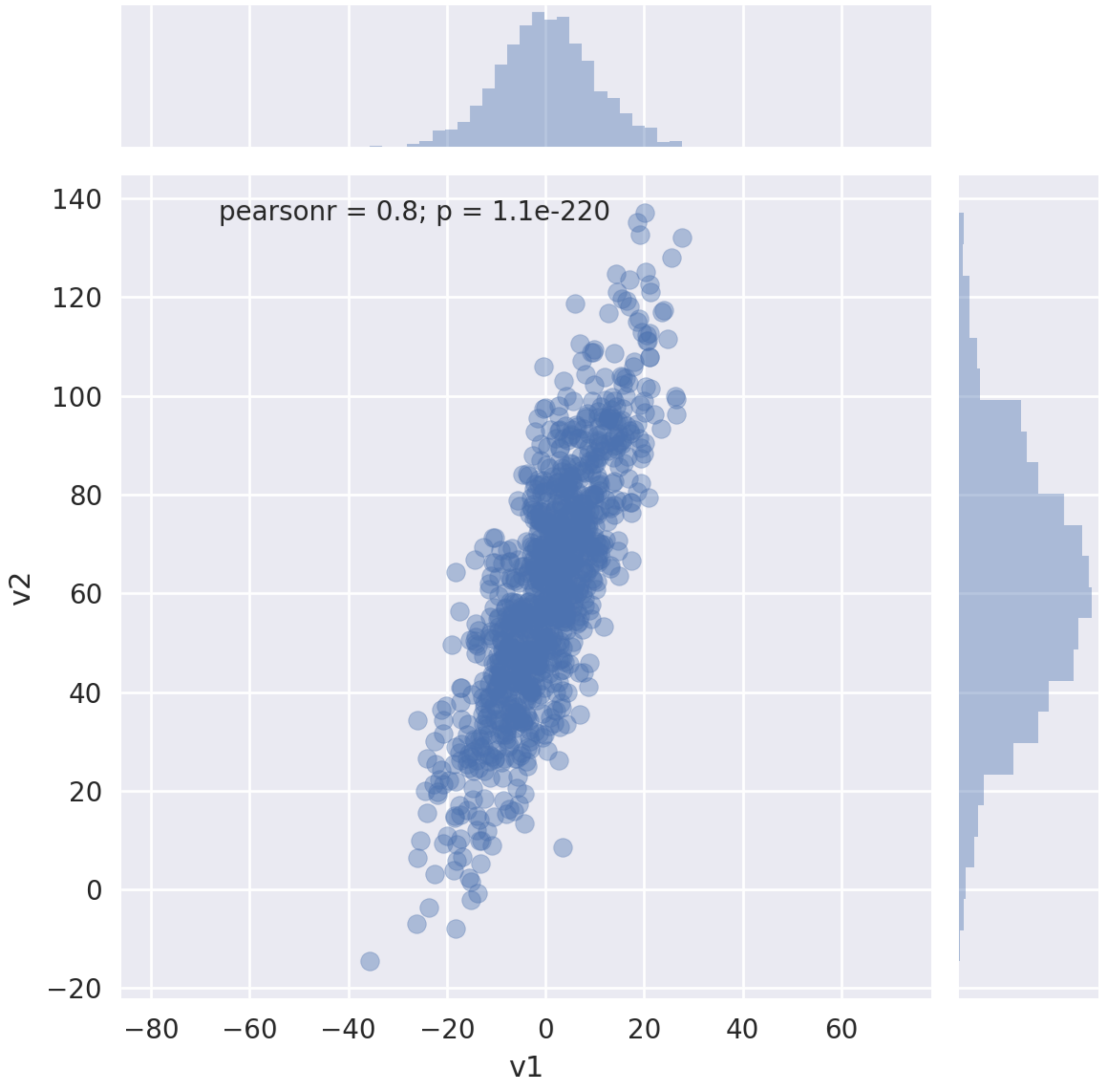
```
In [20]: sns.jointplot(v1, v2, alpha=0.4);
```

Figure 4



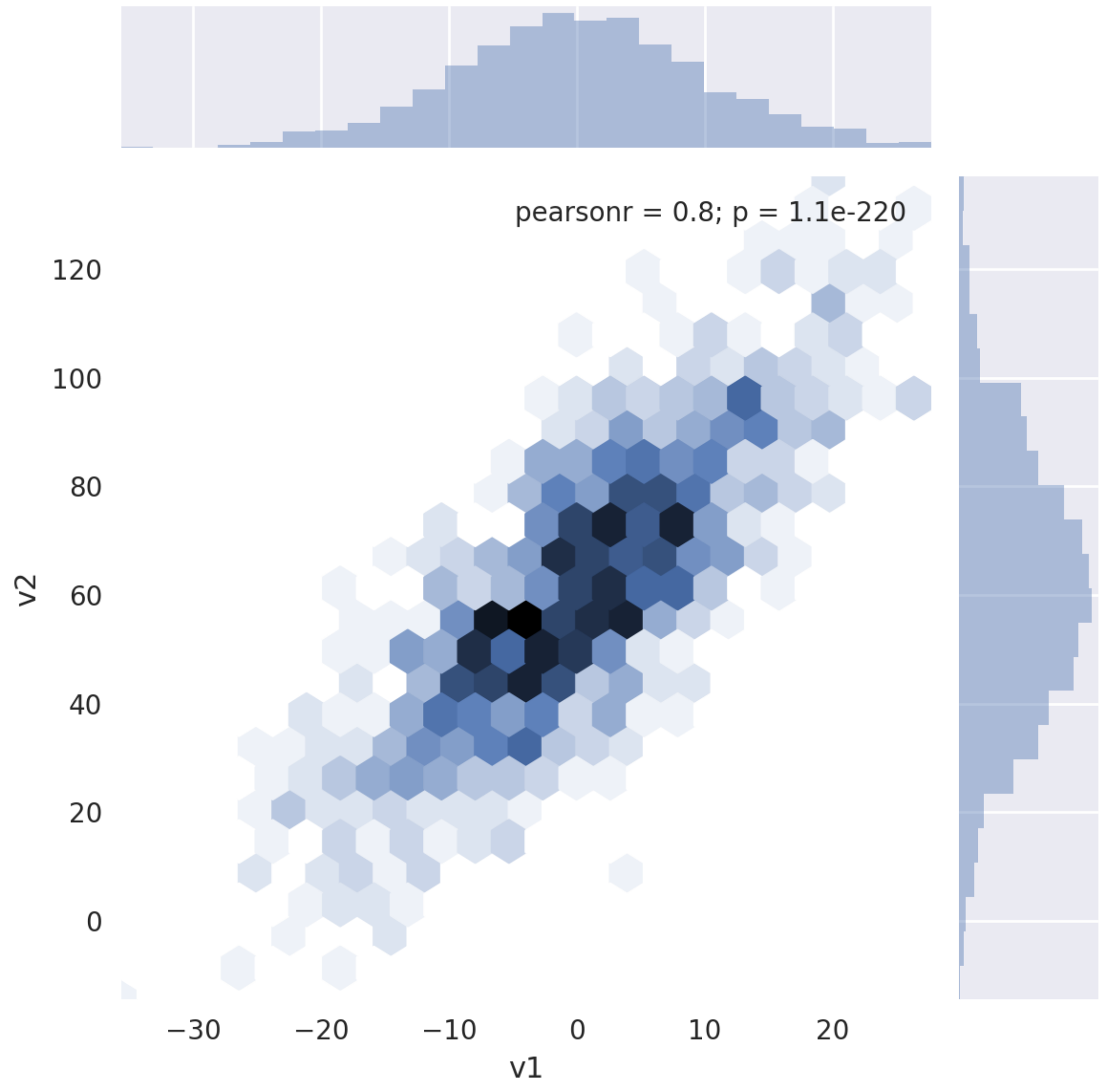
```
In [21]: grid = sns.jointplot(v1, v2, alpha=0.4);
         grid.ax_joint.set_aspect('equal')
```

Figure 5


pearsonr = 0.8; p = 1.1e-220
140
120
100
80
60
40
20
0
−20
v2
−80
−60
−40
−20
0
20
40
60
v1

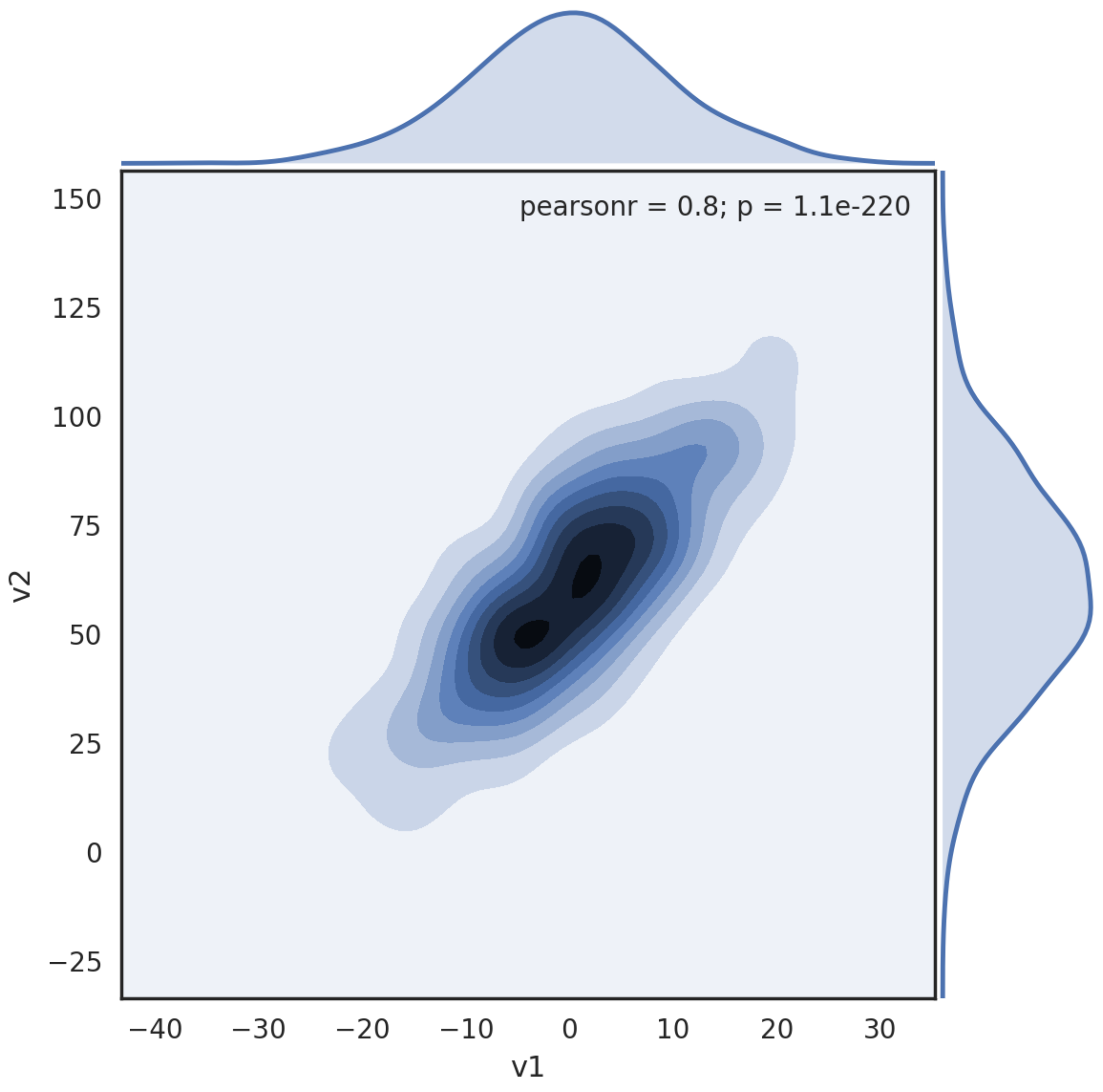
```
In [22]: sns.jointplot(v1, v2, kind='hex');
```

Figure 6



In [23]:
```
# set the seaborn style for all the following plots
sns.set_style('white')

sns.jointplot(v1, v2, kind='kde', space=0);
```

Figure 7


pearsonr = 0.8; p = 1.1e-220
150
125
100
75
50
25
0
−25
v2
−40
−30
−20
−10
0
10
20
30
v1

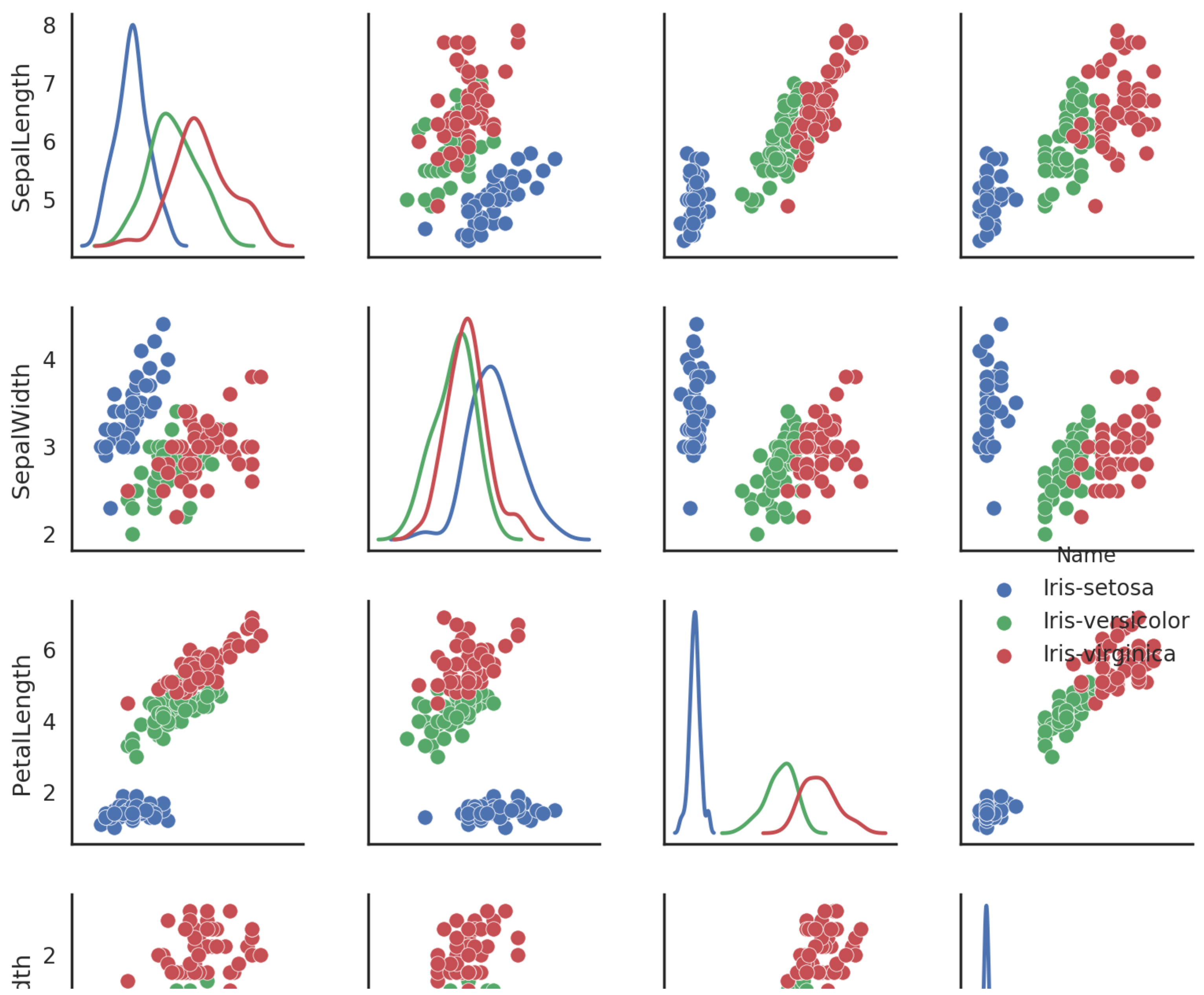
In [24]:
```
iris = pd.read_csv('iris.csv')
iris.head()
```

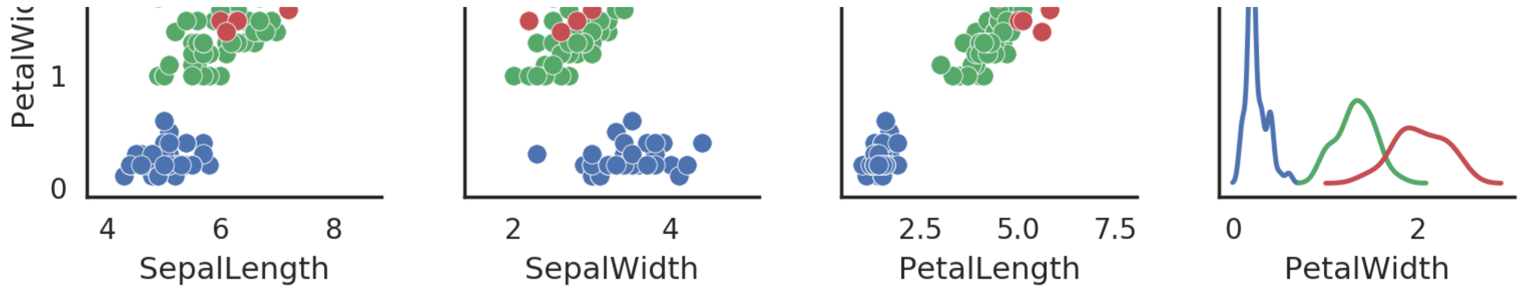
Out[24]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Name
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [25]:
```
sns.pairplot(iris, hue='Name', diag_kind='kde', size=2);
```

Figure 8





```
In [26]: plt.figure(figsize=(8,6))
         plt.subplot(121)
         sns.swarmplot('Name', 'PetalLength', data=iris);
         plt.subplot(122)
         sns.violinplot('Name', 'PetalLength', data=iris);
```

Figure 9